PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Vizualizace algoritmů třídění

2011 Lenka Porketová

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně.

V Olomouci dne 16. srpna 2011 .....

Anotace

Cílem této bakalářské práce bylo vytvořit aplikaci, která demonstruje činnost algoritmů vnitřního třídění. Aplikace byla implementovaná v prostředí Microsoft Visual C# 2010. V textu práce je první kapitola věnovaná tématu algoritmus. Následující kapitola je zaměřena na třídění, především na popis vybraných algoritmů vnitřního třídění. Obsahem třetí kapitoly je uživatelská příručka a v poslední kapitole se nachází programátorská dokumentace.

Děkuji vedoucímu mé bakalářské práce, panu RNDr. Arnoštu Večerkovi, za ochotu, cenné rady a připomínky, které mi poskytl při tvorbě této práce. Také bych chtěla poděkovat rodině a přátelům za podporu během studia.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Jako téma své bakalářské práce jsem si vybrala Vizualizaci algoritmů třídění. Algoritmy třídění mě zaujaly už při studiu algoritmické matematiky. Chtěla bych si proto o nich rozšířit své znalosti a vytvořit aplikaci, která bude demonstrovat činnost algoritmů třídění.

V každém kroku třídění by tedy mělo být přehledně znázorněno, co se provádí. To znamená, který prvek se s kterým srovnává a co se po srovnání provede, zda prvky zůstanou na místě nebo se prvky vymění nebo se některý prvek přesune na jiné místo. Každý krok třídění by měl být rovněž doprovázen slovním popisem.

Implementovány budou následující algoritmy vnitřního třídění: třídění přímým vkládáním (Insert sort), třídění binárním vkládáním (Binary insert sort), Shellovo třídění (Shell sort), třídění přímou výměnou (Bubble sort), třídění přetřásáním (Shaker sort), rychlé třídění výměnou (Quicksort), třídění přímým výběrem (Select sort) a třídění haldou (Heap sort).

Aplikace by měla pomoci pochopit studentům nebo případným zájemcům o algoritmy třídění, jak který algoritmus funguje.

První dvě kapitoly budou zaměřeny na teorii. V dalších dvou se budou zabývat vytvořenou aplikací.

1. Algoritmus

V této kapitole si objasníme původ slova algoritmus [6], [10]. Řekneme si, co je to algoritmus a jaké by měl mít vlastnosti [7]. Dále si ukážeme několik základních způsobů, kterými můžeme algoritmy zapsat [8], [4]. Vysvětlíme si pojem složitost algoritmu [2], [3] a budeme se také zabývat nejznámějšími druhy algoritmů [9].

1.1. Původ slova algoritmus

Slovo „algoritmus“ je odvozeno od jména významného perského matematika, kterým byl Abu Ja'far Muhammad ibn Musa al-Khwarizmi (asi 780 - 850 n. l.). Al-Khwarizmi se zabýval především algebrou a řešením lineárních a kvadratických rovnic. Kolem roku 825 napsal spis o indickém početním systému, v němž ukazoval jak jednoduše sčítat, odčítat, násobit a dělit. Tento spis byl ve 12. století přeložen do latiny jako *Algoritmi de numero Indorum* (česky *Algoritmi o číslech od Indů*). Zlatinizované jméno autora Algoritmi se stalo základem slova algoritmus, které se používalo pro popis různých matematických postupů. Teprve asi od 20. století se termín algoritmus používá v dnešním významu.

Obrázek 1. Abu Ja'far Muhammad ibn Musa al-Khwarizmi.

1.2. Algoritmus a jeho vlastnosti

Algoritmus je jednoduše řečeno návod, jak provést určitou činnost. S algoritmy se běžně setkáváme v životě, aniž bychom si to uvědomovali. Když vypisujeme poštovní poukázku, posíláme SMS zprávy z mobilního telefonu nebo si ráno vaříme kávu, postupujeme podle určitého návodu.

Formálněji lze `algoritmus` charakterizovat jako předpis, který pro libovolná povolená vstupní data dává po konečném počtu kroků požadovaný výsledek, přičemž musí být splněny následující vlastnosti algoritmu:

- `Konečnost` - algoritmus poskytuje požadovaný výsledek v „rozumném“ čase, tj. v čase, kdy výsledek výpočtu ještě můžeme smysluplně využít.
- `Hromadnost` - algoritmus slouží k řešení celé třídy (skupiny) úloh, vzájemně se lišících pouze vstupními údaji.
- `Jednoznačnost (determinovanost)` - v každém kroku algoritmu musí být jednoznačně určeno, co je výsledkem tohoto kroku a jak má algoritmus dále pokračovat.
- `Opakovatelnost` - při stejných hodnotách vstupních dat dospěje algoritmus ke stejnému výsledku.
- `Rezultativnost` - algoritmus vede ke správnému výsledku.

1.3. Zápis algoritmu

Umět správně zapsat algoritmus je důležité pro jeho další použití a úpravu. Způsobů jak lze algoritmus vyjádřit existuje mnoho. Volba vhodného způsobu zápisu závisí na charakteru řešené úlohy. Uvedeme si některé často používané.

`Slovní popis` je určen pro skupinu lidí, kteří nemají programátorské znalosti. Jednotlivé kroky algoritmu jsou vyjádřeny větami v přirozeném jazyce.

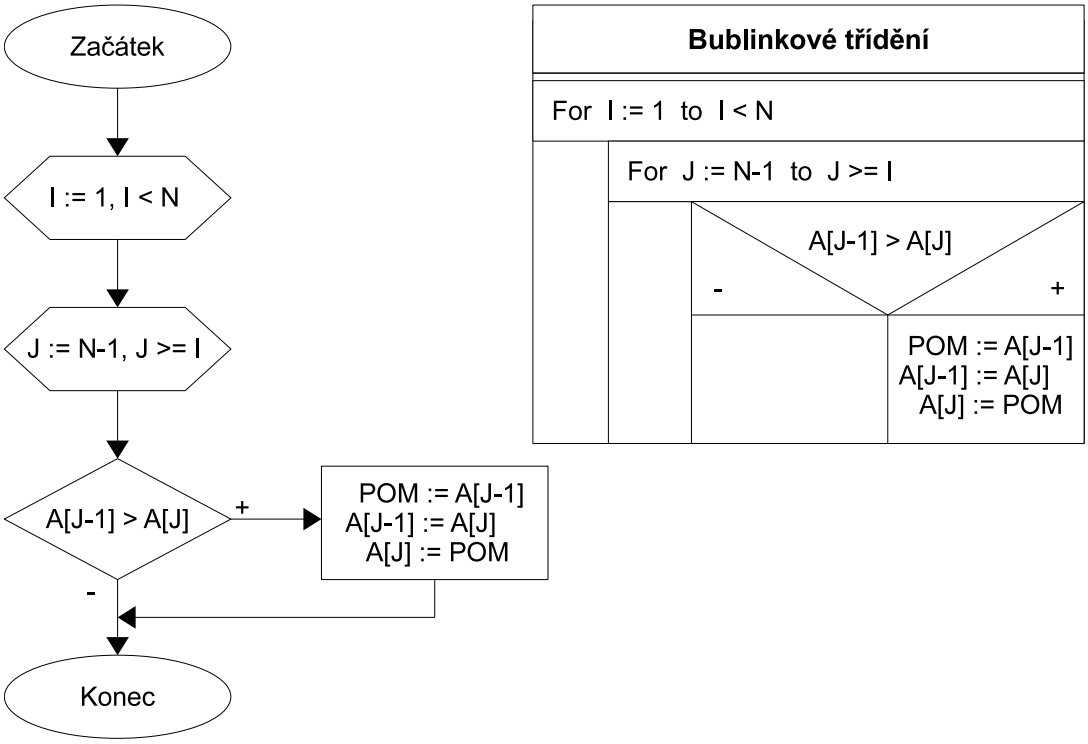
`Pseudokód` je jasný a srozumitelný popis algoritmu, který používá obecně známé strukturní konvence programovacích jazyků, ale nezabývá se implementačními detaily. Můžeme tak zapisovat algoritmy ve formě srozumitelné všem programátorům bez ohledu na to, jaký programovací jazyk znají.

`Programovací jazyk` je prostředek určený pro zápis algoritmů, které mohou být provedeny na počítači. Zápis algoritmu v programovacím jazyku označujeme jako *program*. Každý programovací jazyk má svoji syntaxi[1] a sémantiku[2].

`Vývojový diagram` se skládá z grafických symbolů, jejichž tvar a význam je dán českou státní normou ČSN ISO 5807 z roku 1996. Do jednotlivých symbolů se vpisují slovní nebo symbolickou formou upřesňující údaje. Vývojový diagram procházíme směrem shora dolů.

[1] Syntaxe programovacího jazyka popisuje formální strukturu programu.
[2] Sémantika programovacího jazyka určuje logický význam jednotlivých výrazů jazyka.

`Strukturogram` graficky znázorňuje strukturu algoritmu. Je tvořen obdélníkem, v němž jsou vypsány kroky, které algoritmus vykonává. Záhlaví strukturogramu obsahuje název algoritmu nebo dílčí operace. Do strukturogramu lze vnořovat další strukturogramy.



Obrázek 2. Vývojový diagram (vlevo) a strukturogram (vpravo) bublinkového třídění.

`Rozhodovací tabulka` přehledně zobrazuje variace činností, které při kombinaci všech možných podmínek mohou nastat v souvislosti s řešením problému. Rozhodovací tabulka bývá rozdělena do 4 kvadrantů, jejichž náplň zachycuje tabulka 1.

Záhlaví rozhodovací tabulky	**Záhlaví pravidel**
1. Seznam podmínek, které ovlivňují řešení problému a stanovují jeho možné varianty.	3. Kombinace podmínek, které se mohou vyskytnout.
2. Seznam činností, které je třeba vykonat v rámci všech variant řešení daného problému.	4. Činnosti, které je nutné provést při jednotlivých kombinacích podmínek.

Tabulka 1. Základní tvar rozhodovací tabulky.

1.4. Složitost algoritmu

Složitost algoritmu můžeme posuzovat podle doby prováděného algoritmu (časové složitosti) a rozsahu použité operační paměti (paměťové složitosti).

`Časovou složitost` algoritmu definujeme jako funkci, která každé množině vstupních dat přiřazuje počet elementárních operací nutných k vykonání daného algoritmu.

Poznámka. Časovou složitost nemůžeme vyjadřovat v jednotkách času, protože skutečná doba výpočtu programu nezávisí jen na algoritmu, ale také na zvoleném programovacím jazyce, překladači, procesoru počítače apod.

`Paměťovou složitost` algoritmu definujeme jako funkci, která každé množině vstupních dat přiřazuje počet paměťových míst potřebných k vykonání daného algoritmu.

Dále se budeme zabývat už jen časovou složitostí, která je hlavním kritériem pro posuzování kvality algoritmů.

Časovou složitost můžeme určit z hlediska:

- `Nejhoršího případu` (O) - udává maximální počet operací provedených algoritmem pro libovolná vstupní data. S časovou složitostí v nejhorším případě se pracuje nejčastěji.
- `Průměrného případu` (Θ) - udává průměrný (očekávaný) počet operací provedených algoritmem pro libovolná vstupní data. Časová složitost v průměrném případě charakterizuje algoritmus lépe, ale je těžší ji odvodit.
- `Nejlepšího případu` (Ω) - udává minimální počet operací provedených algoritmem pro libovolná vstupní data. Prakticky se časová složitost v nejlepším případě nepoužívá.

Vyjádřit přesně časovou složitost algoritmu je většinou komplikované. Obvykle proto stačí určit tzv. *asymptotickou složitost*. U asymptotické složitosti zanedbáváme pomaleji rostoucí členy a konstanty. Např. u algoritmu s časovou složitostí $n^2 + 5n + 4$ uvedeme jeho časovou složitost jen jako $O(n^2)$.

Následující složitosti jsou seřazeny od nejrychlejší k nejpomalejší:

- $O(\log(n))$ - logaritmická složitost,
- $O(n)$ - lineární složitost,
- $O(n \log(n))$ - lineárně-logaritmická složitost,
- $O(n^2)$ - kvadratická složitost,
- $O(2^n)$ - exponenciální složitost.

Za prakticky použitelné považujeme algoritmy s polynomickou časovou složitostí. Jsou to algoritmy, které mají časovou složitost vyjádřenou přímo polynomem nebo existuje polynom, který funkci časové složitosti shora ohraničuje. Z výše uvedených složitostí jsou to všechny kromě exponenciální složitosti.

1.5. Dělení algoritmů

Algoritmy můžeme klasifikovat různými způsoby. Podle způsobu implementace rozdělujeme algoritmy do těchto skupin:

`Rekurzivní a iterativní algoritmy`. Rekurzivní algoritmy volají samy sebe dokud není dosažena ukončovací podmínka. V každém kroku rekurze musí dojít ke zjednodušení problému. Iterativní algoritmy provádí opakovaně (v cyklu) stejnou úlohu nad měnící se množinou dat.

`Deterministické a nedeterministické algoritmy`. Deterministické algoritmy mají v každém svém kroku právě jednu možnost jak pokračovat. Nedeterministické algoritmy mohou v některých krocích volit z několika možností dalších kroků. Nedeterministický algoritmus může pro stejná vstupní data dávat rozdílné výsledky.

`Paralelní a sériové algoritmy`. Paralelní algoritmy vykonávají všechny kroky zároveň (ve více vláknech). Sériové algoritmy tyto kroky provádí v sérii (jeden po druhém).

`Přesné a přibližné algoritmy`. Přesné řešení některých úloh trvá příliš dlouho nebo je nelze vyřešit v přijatelném čase vůbec. V takových případech je často postačující alespoň přibližné (téměř optimální) řešení.

Další možné rozdělení algoritmů je podle návrhového paradigmatu:

`Algoritmy typu rozděl a panuj` mají zpravidla tři části. Nejprve rozloží úlohu na několik podúloh stejného typu, ale menšího rozsahu. Potom vyřeší podúlohy rekurzivně nebo přímo, je-li podúloha již dostatečně jednoduchá, a nakonec řešení podúloh sjednotí do řešení původní úlohy.

`Algoritmy dynamického programování` postupně řeší části problému od nejjednodušších po složitější, přičemž využívají výsledky již vyřešených jednodušších podproblémů.

`Hladové algoritmy` vybírají vždy tu možnost, která se v dané chvíli jeví jako nejlepší. Nikdy se nevrací ve volbě zpět.

`Redukce` převádí řešený problém na jiný, ekvivalentní problém, který umíme řešit efektivněji.

`Pravděpodobnostní algoritmy` (někdy označované jako `probabilistické`) provádějí některá rozhodnutí náhodně nebo pseudonáhodně.

`Heuristické algoritmy` většinou nevracejí přesná řešení, ale pouze nějaké vhodné přiblížení. Obvykle vychází z intuitivní myšlenky, jak danou úlohu řešit. Často máme pro stejnou úlohu více heuristických algoritmů, které se liší svou komplikovaností, časovou složitostí a přesností výsledků.

2. Třídění

V úvodní podkapitole si objasníme význam pojmu třídění a uvedeme si dvě odlišné skupiny algoritmů třídění. Další podkapitoly budou zaměřeny na popis vybraných algoritmů vnitřního třídění. Při psaní této kapitoly jsem vycházela z materiálů [1], [2], [3], [5] a [7].

2.1. Úvod do třídění

Třídění je činnost se kterou se každý z nás v životě setkal. Abecedně jsou uspořádány knihy v knihovně, podle počtu bodů jsou seřazeny výsledky uchazečů u přijímacího řízení atd.

Obecně tedy pod pojmem `třídění` rozumíme proces, při kterém jsou prvky dané posloupnosti uspořádány vzestupně nebo sestupně.

Nechť máme posloupnost $A = \{a_1, a_2, \ldots, a_n\}$, kde n je počet prvků. Pak posloupnost A je:

- *vzestupně setříděná*, když pro $\forall a_i \in A \Rightarrow a_i \leq a_{i+1}, i \in \langle 0, n-2 \rangle$,
- *sestupně setříděná*, když pro $\forall a_i \in A \Rightarrow a_i \geq a_{i+1}, i \in \langle 0, n-2 \rangle$.

Pokud máme data setříděná, lze nad nimi provádět řadu operací (např. vyhledávání) efektivněji než nad nesetříděnými.

Algoritmy třídění můžeme rozdělit do dvou hlavních skupin, a to na algoritmy vnitřního třídění a algoritmy vnějšího třídění.

`Algoritmy vnitřního třídění` lze použít v případě, kdy předem známe počet prvků tříděné posloupnosti. Během třídění jsou všechny prvky uloženy ve vnitřní paměti počítače, kde k nim můžeme přistupovat v libovolném pořadí.

`Algoritmy vnějšího třídění` naopak slouží pro případ, kdy je tříděných prvků příliš mnoho a do vnitřní paměti se nevejdou. Prvky jsou proto uloženy v souborech na vnější paměti, odkud jsou průběžně v malým počtech přesouvány do vnitřní paměti. Celé třídění je tedy založeno na opakovaném čtení a vytváření souborů.

Podrobněji se budeme zabývat už jen algoritmy vnitřního třídění, které můžeme rozdělit, podle toho z které metody vychází, do tří kategorií a to na:

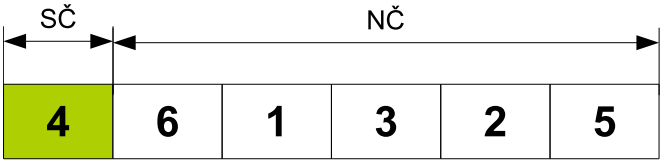
- `Třídění vkládáním` - bereme jednotlivé prvky třídění posloupnosti a vkládáme je na správné místo do setříděné části.
- `Třídění výměnou` - hledáme dvojici prvků, která je ve špatném pořadí a prvky v ní vzájemně vyměníme.
- `Třídění výběrem` - vybíráme prvky z nesetříděné části a přidáváme je k setříděné části.

2.2. Třídění vkládáním

Metodu vkládání využívají následující algoritmy:

2.2.1. Třídění přímým vkládáním

Třídění přímým vkládáním (angl. Insert sort) rozděluje pole o n prvcích na dvě části: setříděnou část (SČ), která je první a nesetříděnou část (NČ). Setříděná část na začátku třídění obsahuje 1. prvek pole a nesetříděná část zbývajících $n-1$ prvků.



Obrázek 3. Počáteční krok algoritmu Insert sort.

V každém kroku třídění vezmeme první prvek nesetříděné části a vložíme ho na příslušné místo do setříděné části. Místo pro vložení můžeme najít tak, že budeme od začátku procházet setříděnou část a porovnávat vkládaný prvek s prvky v setříděné části. Abychom uvolnili místo pro vkládaný prvek, posuneme o jedno místo dozadu všechny prvky počínaje místem vložení. Efektivnější ale je, když procházíme setříděnou část od konce a současně při tom posunujeme prvky v setříděné části o jednu pozici dozadu.

Velikost setříděné části se po každém provedení kroku třídění zvětší o jeden prvek. Třídění bude ukončeno, až bude setříděná část obsahovat všech n prvků.

Časová složitost třídění přímým vkládáním je $\Theta(n^2)$.

2.2.2. Třídění binárním vkládáním

Třídění binárním vkládáním (angl. Binary insert sort) se liší od třídění přímým vkládáním pouze v hledání místa pro vkládaný prvek. Místo pro vkládaný prvek najdeme tak, že vkládaný prvek porovnáme s prostředním prvkem setříděné části. Porovnáním zjistíme, zda vkládaný prvek patří do části vlevo nebo vpravo od prostředního prvku. Tento postup, který se nazývá binární vyhledávání, opakujeme tak dlouho, dokud nenajdeme správné místo pro vložení prvku.

I když počet srovnání u třídění binárním vkládáním může být nižší (záleží na uspořádání prvků tříděné posloupnosti) než u třídění přímým vkládáním, počet přesunů bude prakticky stejný.

Třídění binárním vkládáním má tedy také časovou složitost $\Theta(n^2)$.

2.2.3. Shellovo třídění

Shellovo třídění (angl. Shell sort) je také založeno na třídění přímým vkládáním. Posloupnost s n prvky rozdělíme na h podposloupností (kde $h > 1$) tak, že budou obsahovat prvky, které jsou od sebe vzdáleny o h prvků. Tyto podposloupnosti setřídíme tříděním přímou výměnou.

Pro větší hodnotu h je lepší složitost třídění, u menší hodnoty h se zase prvky dostanou blíže ke své cílové pozici. Shellovo třídění proto provádí dělení tříděné posloupnosti na podposloupnosti vícekrát a hodnotu h postupně snižuje. Hodnotám h se říká kroky Shellova třídění.

Optimální posloupnost hodnot $h_i, h_{i-1}, \ldots, h_2, h_1$ se vytvoří předpisem

$$h_1 = 1, \qquad h_i = 2 \cdot h_{i-1} + 1 \qquad \text{pro } i > 1.$$

První zvolená hodnota by měla splňovat podmínku $h_i \leq \frac{n}{2}$.

Při Shellově třídění s krokem 1 (běžné třídění přímým vkládáním) se třídí najednou celá posloupnost. Výsledkem je setříděná posloupnost.

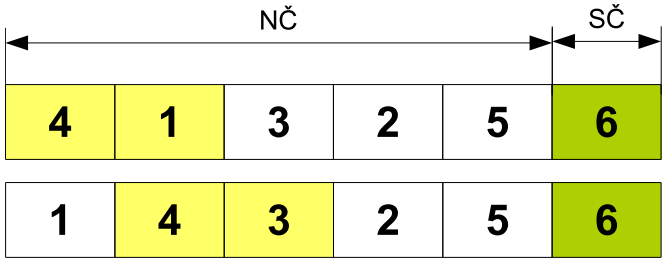
Časová složitost Shellova třídění je $\Theta(n^{1,25})$.

2.3. Třídění výměnou

Na metodě výměny jsou založeny tyto algoritmy:

2.3.1. Třídění přímou výměnou

Třídění přímou výměnou je známé pod názvem bublinkové třídění (angl. Bubble sort). Na začátku třídění považujeme celé pole za nesetříděnou část. Pole s prvky procházíme zleva doprava a srovnáváme vždy dva sousední prvky. Pokud nejsou ve správném pořadí, vyměníme je. Tak se největší prvek nesetříděné části dostane na její konec, tedy před setříděnou část, kterou o tento prvek zvětšíme.



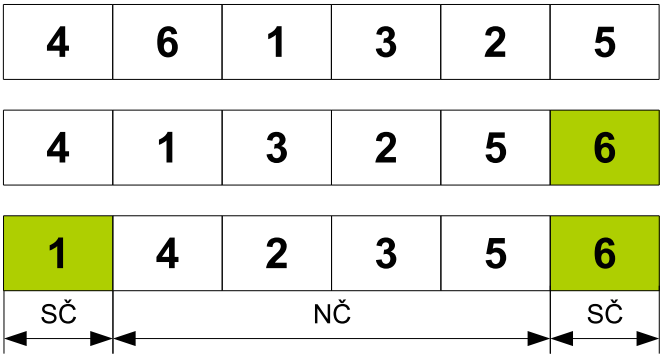
Obrázek 4. Ukázka bublinkového třídění (srovnávané prvky jsou žlutě).

Délka nesetříděné části se po každém průchodu o jeden prvek zmenší, až nakonec bude obsahovat jen dva prvky. Po jejich srovnání a případné výměně bude třídění dokončeno.

Třídění přímou výměnou má časovou složitost $\Theta(n^2)$.

2.3.2. Třídění přetřásáním

Třídění přetřásáním (angl. Shaker sort) je modifikované bublinkové třídění, které střídá směry procházení nesetříděné části zleva doprava a zprava doleva. Setříděná část tak vzniká na obou koncích pole.



Obrázek 5. Stav po prvních dvou průchodech algoritmem Shaker sort.

Třídění přetřásáním je rychlejší, než bublinkové třídění, protože provede méně operací srovnání. Počet výměn, ale zůstane stejný, takže časová složitost třídění přetřásáním zůstává $\Theta(n^2)$.

2.3.3. Rychlé třídění výměnou

Rychlé třídění výměnou je známé více pod svým anglickým názvem Quicksort. Můžeme se ale také setkat s označením třídění rozdělováním.

Nejprve určíme prostřední prvek pole (pokud máme sudý počet prvků, je to ten první ze dvou prostředních), tzv. pivot. Označme si ho p. Na první prvek v poli si nastavíme index i a na poslední prvek v poli index j. Potom procházíme pole zleva, zvyšujeme hodnotu indexu i a hledáme prvek větší než je pivot. Následně procházíme pole zprava, snižujeme hodnotu indexu j a hledáme prvek menší než je pivot. Oba nalezené prvky spolu vyměníme.

S výměnami prvků pokračujeme tak dlouho, dokud se oba indexy nepotkají. V té chvíli máme pole rozdělené na dvě části, kde levá část obsahuje prvky menší než je pivot a pravá část prvky větší než je pivot.

Stejný postup rekurzivně aplikujeme na obě části, které vznikly po rozdělení pole. Rekurze končí, je-li předán pouze jeden prvek k setřídění.

Časová složitost rychlého třídění výměnou je v průměrném případě $\Theta(n \log(n))$ a v nejhorším případě $O(n^2)$.

2.4. Třídění výběrem

Metoda výběrem je základem těchto algoritmů:

2.4.1. Třídění přímým výběrem

Třídění přímým výběrem (angl. Select sort) je založeno na výběru nejmenšího prvku z nesetříděné části.

Nesetříděnou část tvoří na začátku třídění celé pole s n prvky. Výběr nejmenšího prvku provedeme tak, že si zapamatujeme pozici prvního prvku v nesetříděné části a tento prvek postupně srovnáváme s dalšími prvky v poli. Pokud najdeme menší prvek, budeme si dále pamatovat jeho pozici. Tímto postupem dosáhneme toho, že na konci pole budeme znát pozici nejmenšího prvku.

Nalezený nejmenší prvek vyměníme s prvním prvkem nesetříděné části a zvětšíme o něj setříděnou část. Pole tedy budeme mít rozdělené na dvě části, kde první bude setříděná část a druhá bude nesetříděná část.

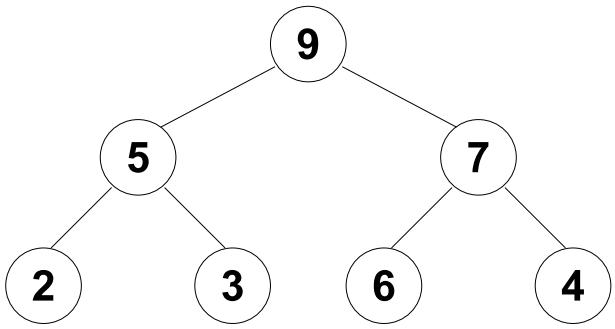
Výběr nejmenšího prvku budeme opakovat tak dlouho, dokud setříděná část nebude obsahovat $n - 1$ prvků.

Třídění přímým výběrem má časovou složitost $\Theta(n^2)$.

2.4.2. Třídění haldou

Třídění haldou (angl. Heap sort) je vylepšený algoritmus třídění přímým výběrem. Vylepšení spočívá v efektivnějším výběru požadovaného prvku, k němuž je zapotřebí datová struktura halda.

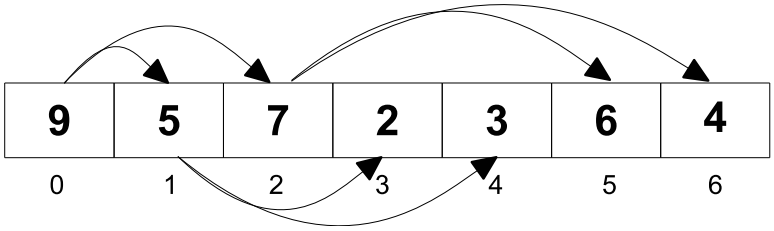
Haldu lze definovat jako vyvážený binární strom[3], který má v každém uzlu právě jeden prvek tříděné posloupnosti. Dále musí halda splňovat podmínku, že prvek v libovolném uzlu je větší nebo stejný, než jsou prvky v jeho následnících, má-li nějaké.



Obrázek 6. Příklad haldy.

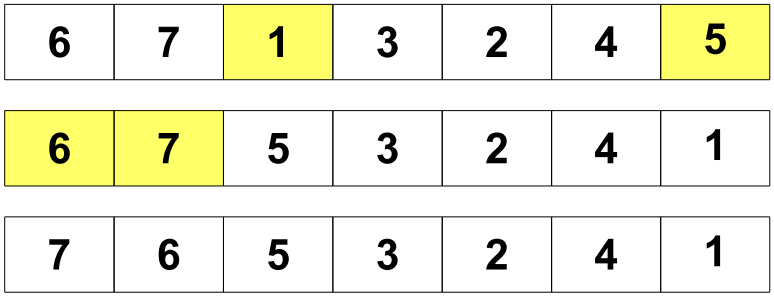
[3] U vyváženého binárního stromu se hloubka jeho listů může lišit nejvýše o 1.

Halda se dá jednoduše implementovat pomocí pole. Pole je běžně indexované od nuly, takže na nulté pozici pole bude kořen haldy. Ten bude mít následníky na první a druhé pozici v poli. Prvek na první pozici v poli bude mít následníky na třetí a čtvrté pozici atd. I-tý prvek v poli bude mít následníky na pozicích $2i+1$ a $2i+2$.



Obrázek 7. Halda z obr. 6 reprezentovaná polem.

Při třídění haldou si tedy nejprve z posloupnosti n prvků, které chceme třídit sestavíme haldu.



Obrázek 8. Sestavení haldy ze zadané posloupnosti.

Celé pole s n prvky budeme na začátku třídění považovat za nesetříděnou část. V rámci každého kroku třídění vybereme poslední prvek haldy (pole) a vyměníme ho s kořenem haldy (prvním prvkem pole). Prvek na poslední pozici v poli, tak bude na správném místě a bude tvořit setříděnou část. Ze zbývajících n-1 prvků nesetříděné části znovu sestavíme haldu. Tento postup budeme opakovat tak dlouho, dokud halda nebude obsahovat jen jeden prvek. Tím proces třídění končí.

Časová složitost třídění haldou je v nejhorším i průměrném případě $\Theta(n \log(n))$.

3. Uživatelská příručka

Tato uživatelská příručka začíná návodem, jak nainstalovat aplikaci `Vizualizace algoritmů třídění`. Dále následuje podrobný popis ovládání aplikace a v závěru kapitoly je uvedena jedna z možností, jak aplikaci odinstalovat.

3.1. Systémové požadavky

Pro běh aplikace je doporučená minimální konfigurace:

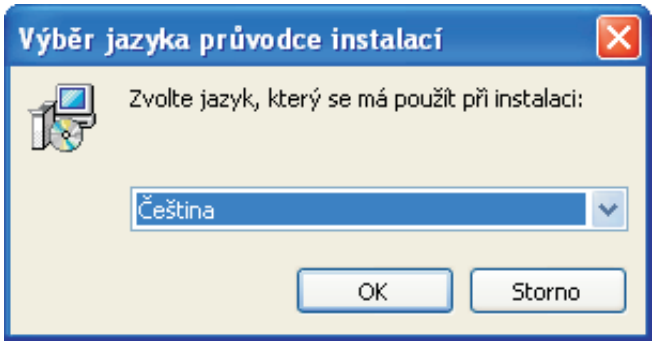
- `operační systém`: Windows XP SP3 a vyšší,
- `běhové prostředí`: Microsoft .NET Framework 4.0,
- `procesor`: 1 GHz,
- `operační paměť`: 512 MB,
- `pevný disk`: přibližně 1,22 MB volného místa, pokud je třeba instalovat Microsoft .NET Framework 4.0, tak asi 850 MB volného místa.

3.2. Instalace aplikace

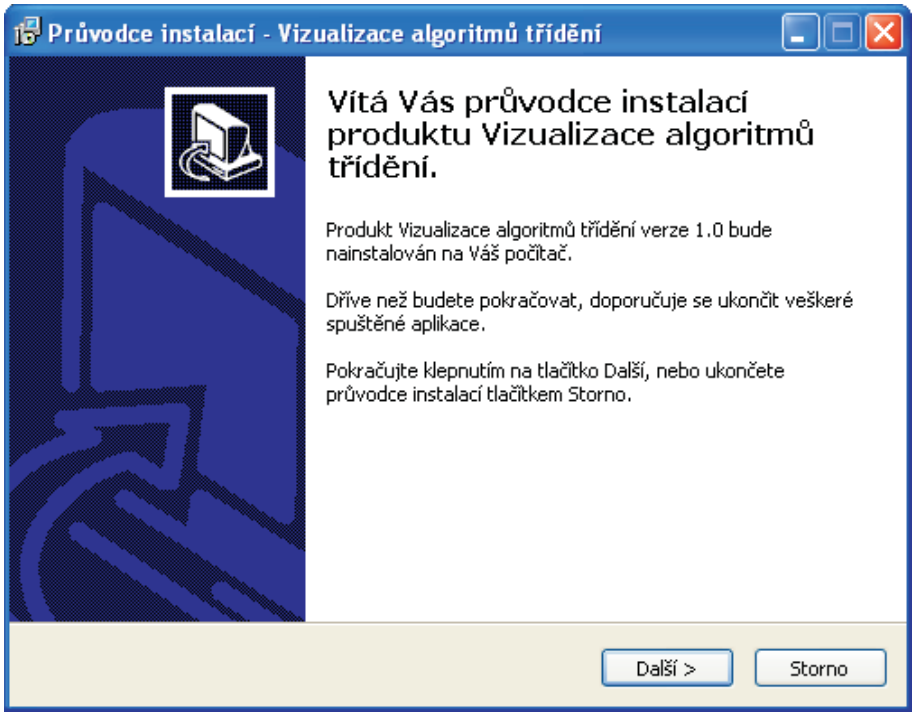
Instalaci aplikace spustíte souborem `VATsetup.exe`, který se nachází v adresáři `bin` na instalačním CD.

Pokud máte nainstalované běhové prostředí Microsoft .NET Framework 4.0 přejděte ke kroku 1. V opačném případě budete na tuto skutečnost upozorněni a instalace bude ukončena. Microsoft .NET Framework 4.0 si můžete nainstalovat z adresáře `install` na instalačním CD spuštěním souboru `dotNET40setup.exe`.

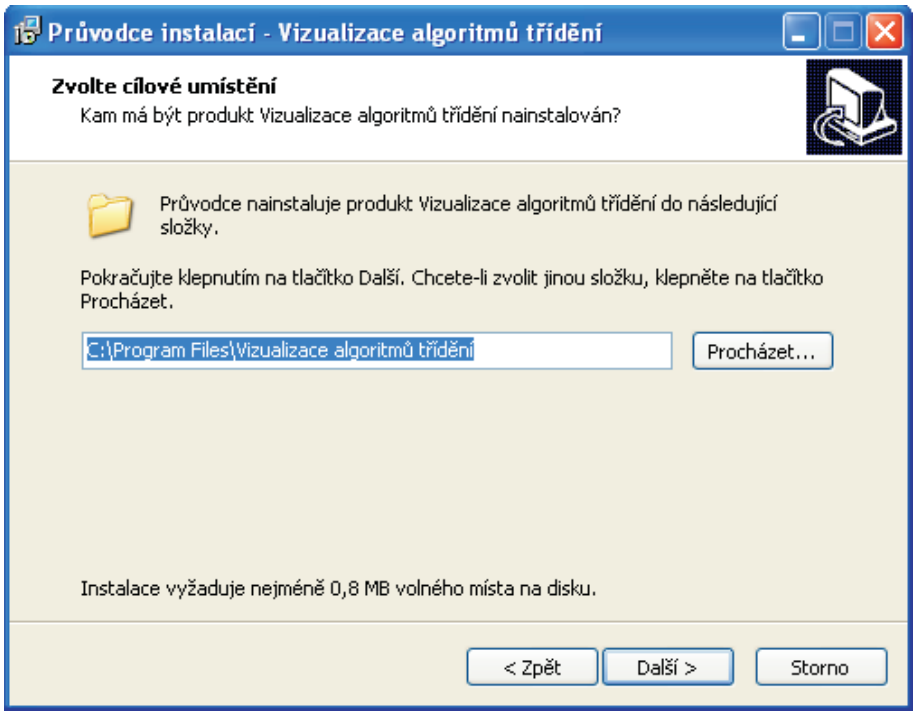
1. Vyberte jazyk průvodce instalací a klikněte myší na tlačítko `OK`.



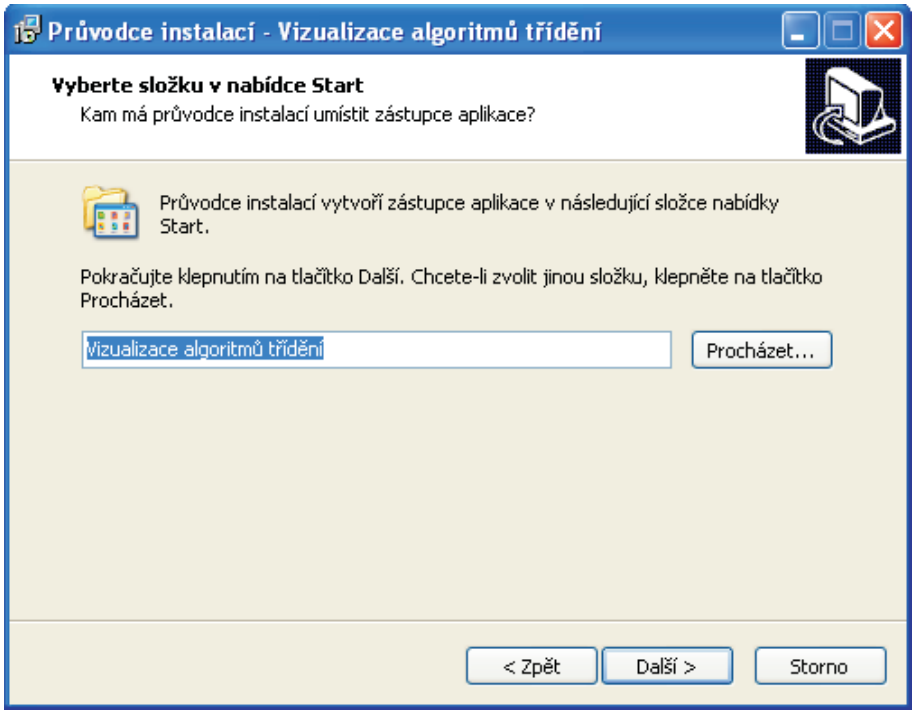
2. Zobrazí se uvítací obrazovka průvodce instalací. Postupujte podle instrukcí průvodce instalací a klikněte na tlačítko `Další`.



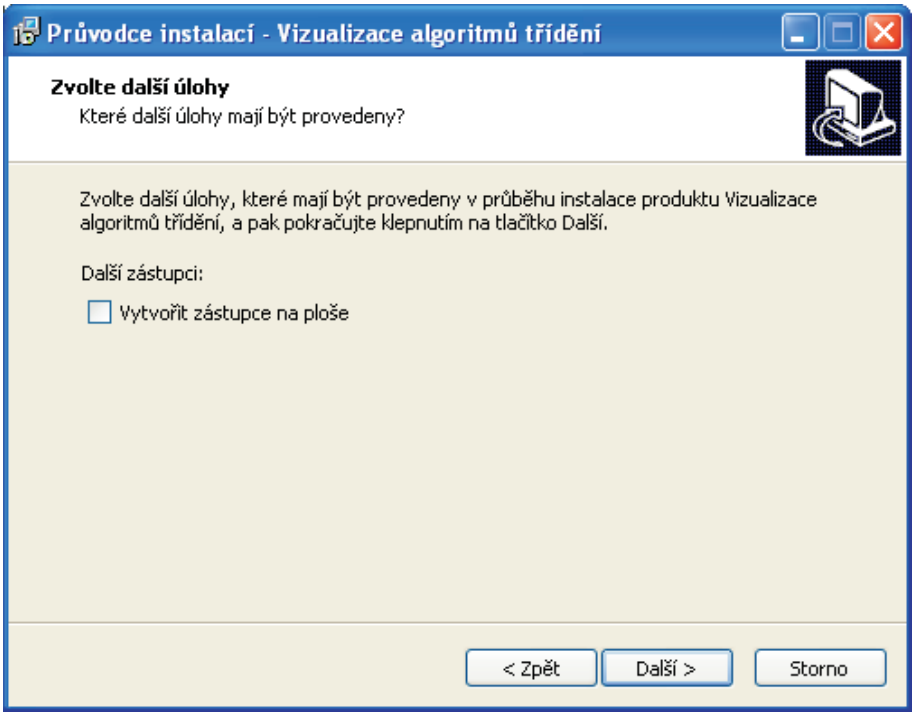
3. Vyberte složku, kam má být aplikace Vizualizace algoritmů třídění nainstalována a klikněte na tlačítko `Další`.



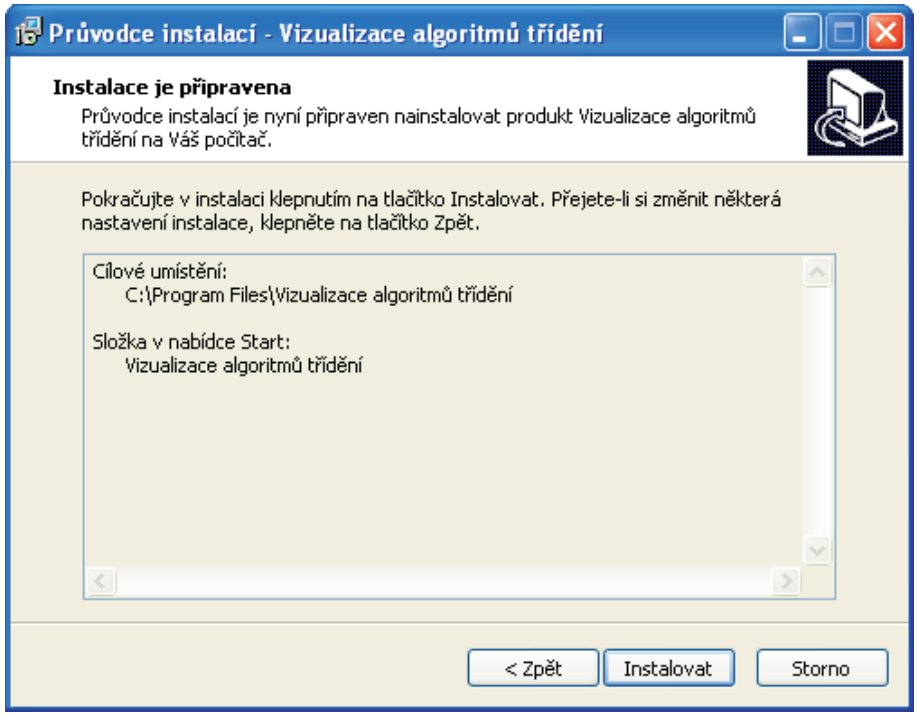
4. Vyberte, do které složky v nabídce Start se má vytvořit zástupce aplikace a klikněte na tlačítko `Další`.



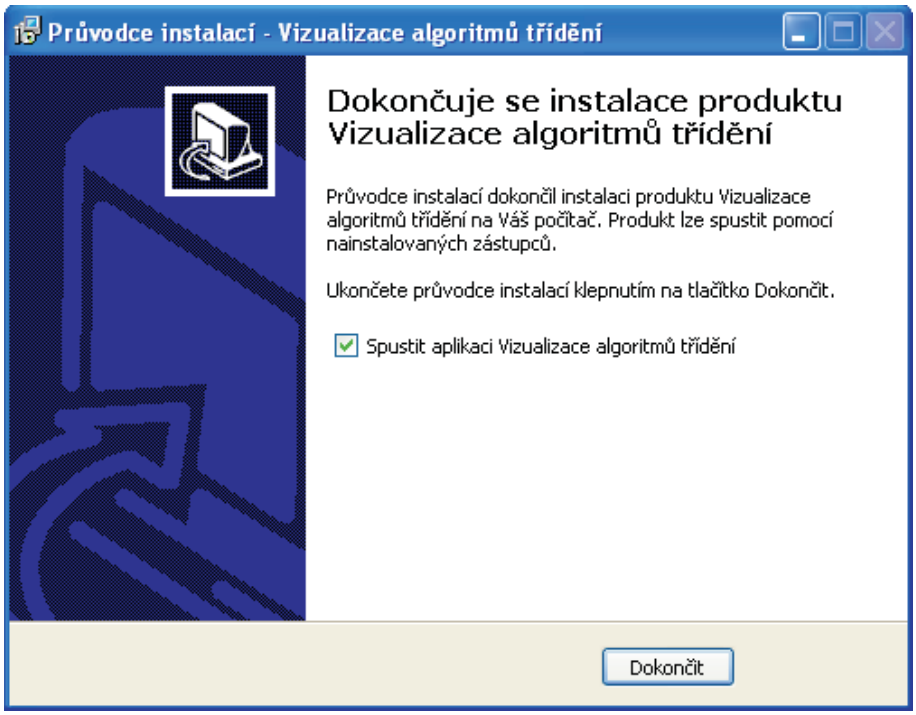
5. Pokud chcete vytvořit zástupce aplikace na ploše, zatrhněte tuto možnost a klikněte na tlačítko `Další`.



6. Nyní máte nastaveno vše potřebné pro instalaci. Kliknutím na tlačítko `Instalovat` zahájíte vlastní instalaci.



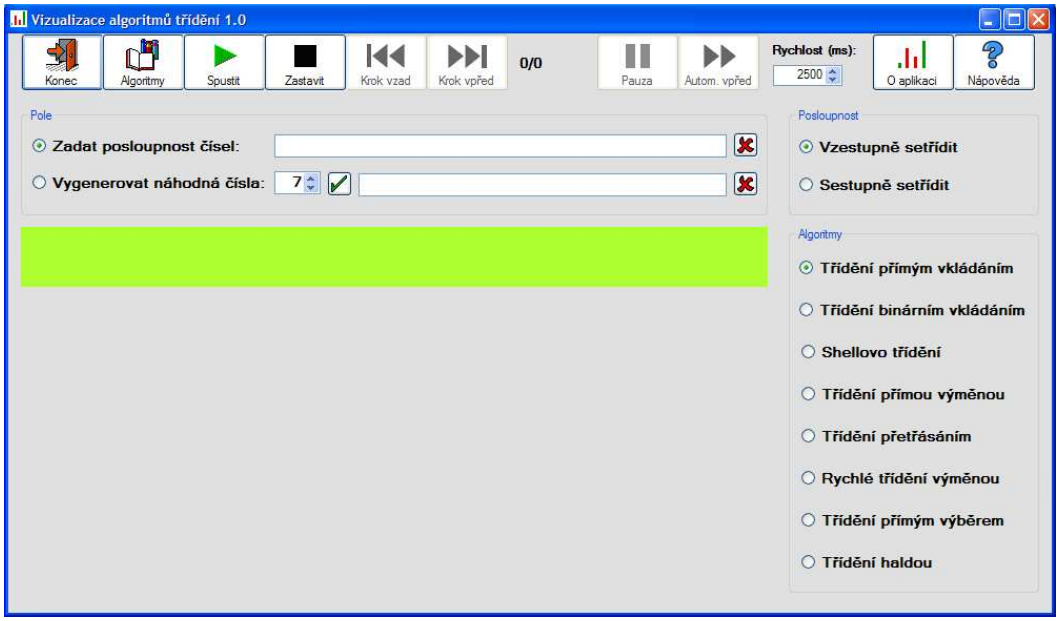
7. Pokud chcete aplikaci po dokončení instalace spustit, nechte tuto možnost zatrženou. Průvodce instalací ukončíte kliknutím na tlačítko `Dokončit`.



3.3. Popis aplikace

3.3.1. Hlavní okno aplikace

Po spuštění aplikace se zobrazí hlavní okno (obr. 9.), které lze rozdělit na několik částí.



Obrázek 9. Hlavní okno aplikace.

V horní části hlavního okna se nachází deset tlačítek. Údaj `0/0` vedle tlačítka `Krok vpřed` udává aktuální krok/celkový počet kroků třídění a `Rychlost (ms)` vedle tlačítka `Autom. vpřed` slouží k nastavení prodlevy mezi jednotlivými kroky třídění u automatického krokování vpřed.

Část vlevo pod tlačítky je určená pro zadávání tříděných čísel.

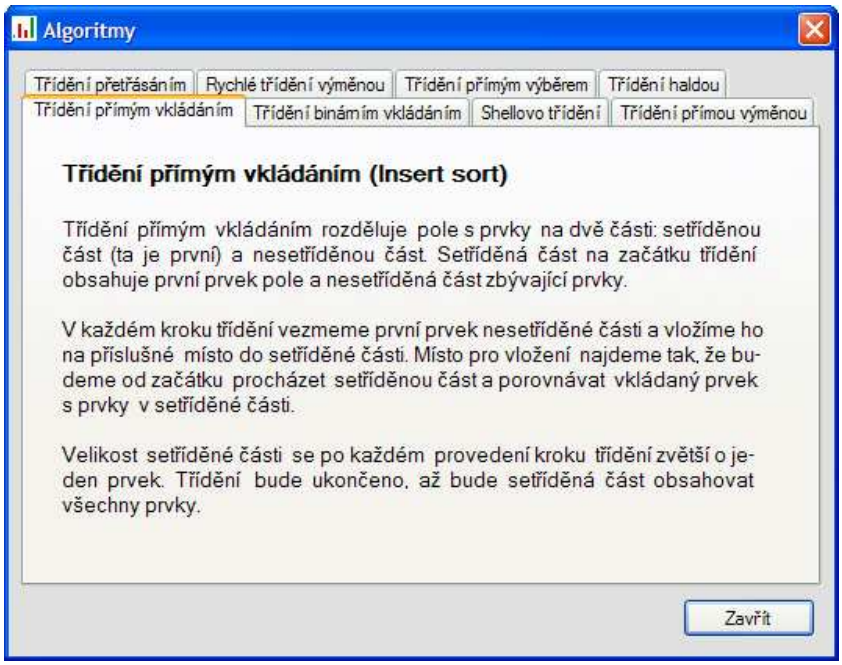
Zda se bude posloupnost čísel třídit vzestupně nebo sestupně je možné zvolit v pravé části hlavního okna, stejně tak jako algoritmus třídění.

Zbývající část hlavního okna je vyhrazena pro demonstraci třídění. Slovní popis jednotlivých kroků třídění se znázorňuje v zeleně vyznačené oblasti.

TIP: Pokud se Vám nezobrazují klávesové zkratky u tlačítek a chcete je zobrazit, stiskněte klávesu ***`Alt`****.*

3.3.2. Okno Algoritmy

Kliknutím na tlačítko `Algoritmy` nebo stisknutím klávesy `G` se zobrazí okno (obr. 10.), kde je na jednotlivých záložkách popsáno, jak který algoritmus třídění funguje. Okno Algoritmy zavřete stisknutím tlačítka `Zavřít`.

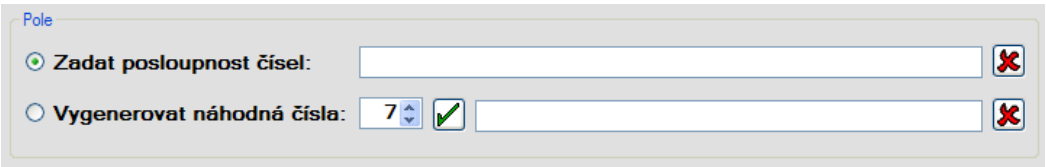


Obrázek 10. Okno Algoritmy.

3.3.3. Zadání čísel ke třídění

Zadat čísla ke třídění můžete dvěma způsoby:

1. Nechat označenou první možnost, tj. `Zadat posloupnost čísel` (obr. 11.) a napsat vedle do textového pole čísla, na kterých chcete demonstrovat činnost vybraného algoritmu třídění. Je třeba zadat minimálně 2 a maximálně 20 čísel z intervalu 0 až 99.



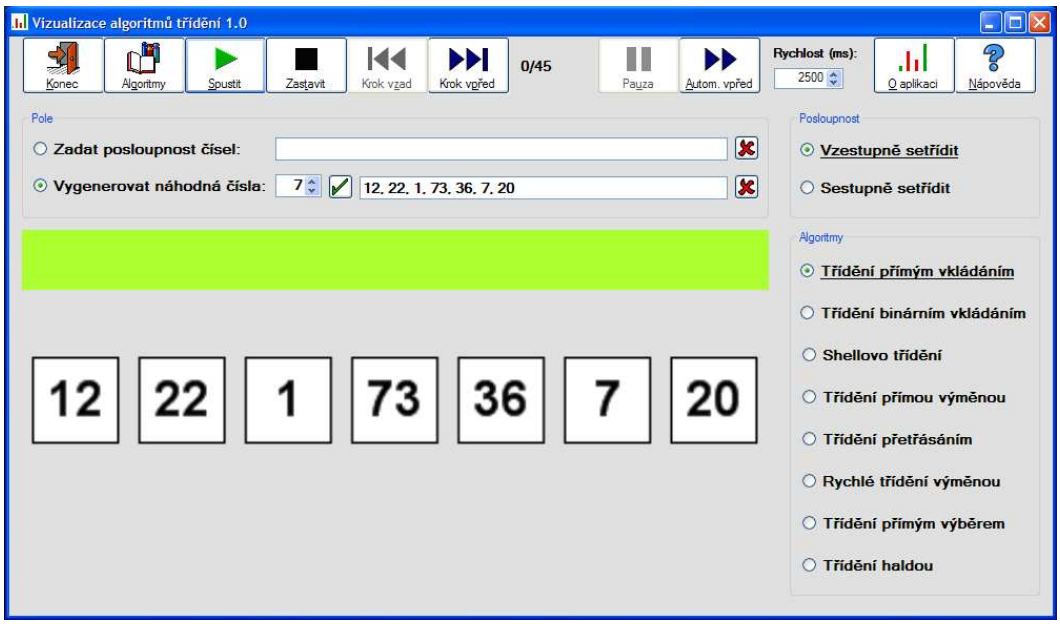
Obrázek 11. Pole pro zadání čísel.

2. Vybrat druhou možnost, tj. `Vygenerovat náhodná čísla`, zvolit počet čísel, které chcete vygenerovat, a stisknout tlačítko `Vygenerovat`. Vygenerovaná čísla se zobrazí v textovém poli.

Tlačítka `Smazat zadaná čísla` a `Smazat vygenerovaná čísla` slouží k vymazání příslušných textových polí.

3.3.4. Spuštění demonstrace třídění

Demonstraci spustíte tlačítkem `Spustit` nebo stisknutím klávesy `S`. Pokud jste korektně nezadali čísla ke třídění, budete na to upozorněni dialogem. V opačném případě se přibližně ve střední části hlavního okna vykreslí čtverce se zadanými čísly (obr. 12.).



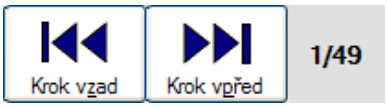
Obrázek 12. Spuštěná demonstrace.

Vybraný způsob třídění (vzestupně nebo sestupně) a algoritmus bude podtržený a tlačítka `Krok vpřed` a `Autom. vpřed` aktivní. Vedle tlačítka `Krok vpřed` se nastaví celkový počet kroků potřebný k setřídění zadaných čísel.

Nyní si můžete buď sami krokovat, jak vybraný algoritmus třídí, nebo spustit automatické krokování vpřed.

Krokování

Ke krokování slouží tlačítka `Krok vzad` a `Krok vpřed` (obr. 13.). Tlačítko `Krok vzad` bude aktivní, jakmile hodnota aktuálního kroku bude větší jak nula.



Obrázek 13. Tlačítka pro krokování.

Místo tlačítka `Krok vzad` lze použít klávesu `Z` a místo tlačítka `Krok vpřed` klávesu `P`.

Automatické krokování vpřed

Automatické krokování vpřed spustíte tlačítkem `Autom. vpřed` (obr. 14.) nebo stiskem klávesy `A`. Jednotlivé kroky třídění se po sobě budou zobrazovat podle nastavené rychlosti. Tlačítkem `Pauza` nebo stiskem klávesy `U` lze automatické krokování vpřed pozastavit.

Obrázek 14. Tlačítka pro automatické krokování vpřed.

Dále se pak třídění spustí opět tlačítkem `Autom. vpřed`.

3.3.5. Zastavení demonstrace třídění

Spuštěnou demonstraci ukončíte kliknutím na tlačítko `Zastavit` (obr. 15.) nebo stiskem klávesy `T`.

Obrázek 15. Tlačítko Zastavit.

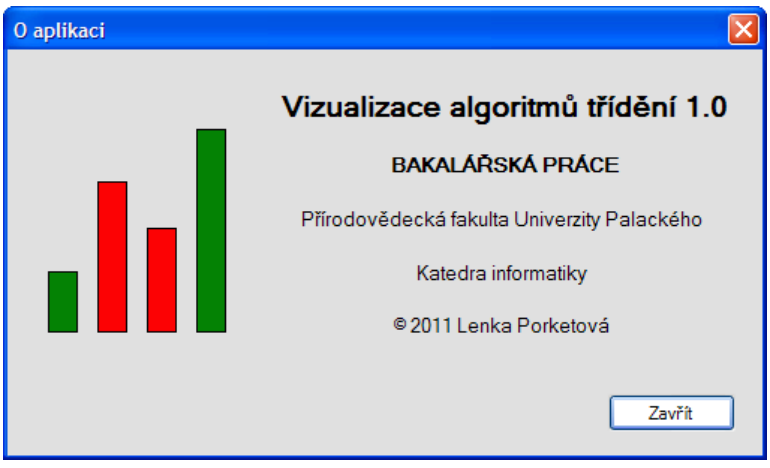
3.3.6. Barevné označení

Pří demonstraci činnosti algoritmů vnitřního třídění se používá následující barevné označení:

nesetříděný prvek	všechny algoritmy třídění
setříděný prvek	všechny algoritmy třídění
srovnávaný prvek	všechny algoritmy třídění
přesunutý/vyměněný prvek	všechny algoritmy třídění
vkládaný prvek	třídění přímým vkládáním
	třídění binárním vkládáním
	Shellovo třídění
neaktivní prvek	Shellovo třídění
dočasně setříděný prvek	Shellovo třídění
zapamatovaná pozice	třídění přímým výběrem
nově nalezená pozice	třídění přímým výběrem
následníci prvku	třídění haldou
pivot	rychlé třídění výměnou

3.3.7. Informace o aplikaci

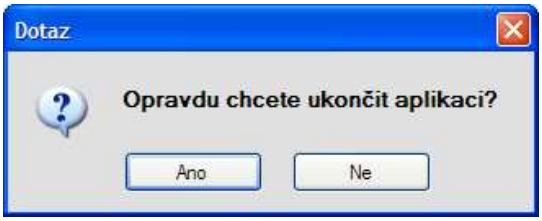
Informace o aplikaci jsou uvedeny v dialogu (obr. 16.), který vyvoláte kliknutím na tlačítko `O aplikaci` nebo stiskem klávesy `O`. Tlačítkem `Zavřít` dialog zavřete.



Obrázek 16. Dialog O aplikaci.

3.3.8. Ukončení aplikace

Aplikaci ukončíte tlačítkem `Konec` nebo stiskem klávesy `K` a potvrzením následujícího dialogu (obr. 17.), že aplikaci chcete opravdu ukončit.



Obrázek 17. Ukončení aplikace.

V případě, že stiskněte tlačítko `Ne`, dialog se zavře a aplikace zůstane spuštěna.

3.4. Odinstalace aplikace

Odinstalovat aplikaci můžete následujícím způsobem:

1. V nabídce `Start > Programy > Vizualizace algoritmů třídění` klikněte na položku `Odinstalovat aplikaci Vizualizace algoritmů třídění`.

2. Dále potvrďte v průvodci odinstalací, že opravdu chcete produkt Vizualizace algoritmů třídění a všechny jeho součásti odinstalovat.

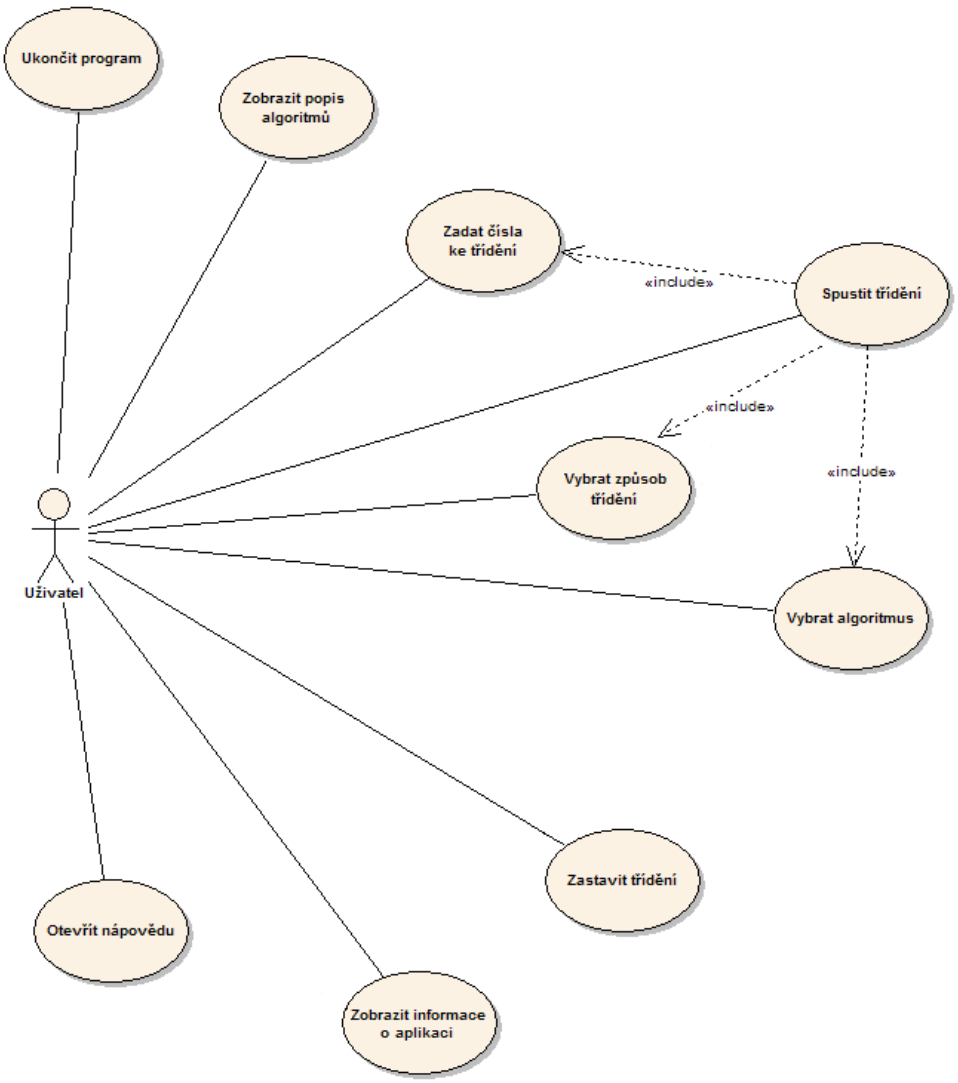
3. Spustí se proces odinstalování, po jehož dokončení budete informování, že produkt Vizualizace algoritmů třídění byl z Vašeho počítače úspěšně odinstalován.

4. Programátorská dokumentace

Aplikace Vizualizace algoritmů třídění byla implementována v programovacím jazyce C# v prostředí Microsoft Visual Studio 2010 Express [11], které využívá Microsoft .NET Framework 4.0.

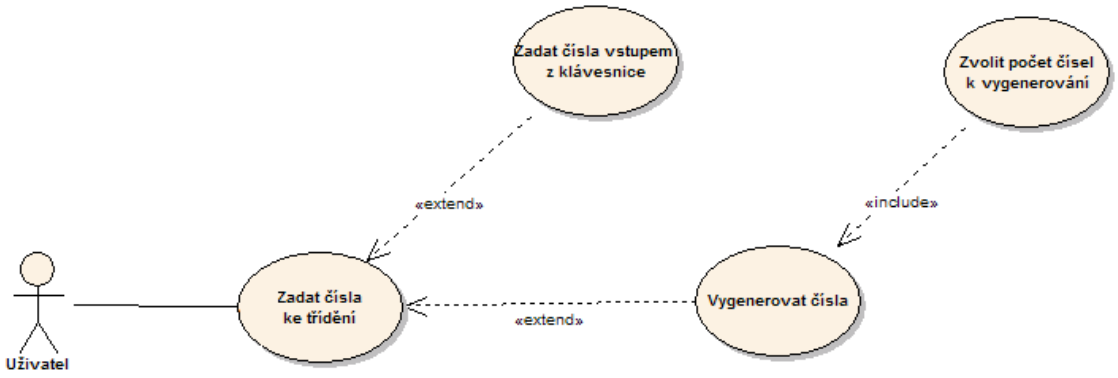
4.1. Use case diagramy

Na obrázku 18. je zachycen základní use case diagram, který nám poskytne představu o jednotlivých funkcích aplikace.



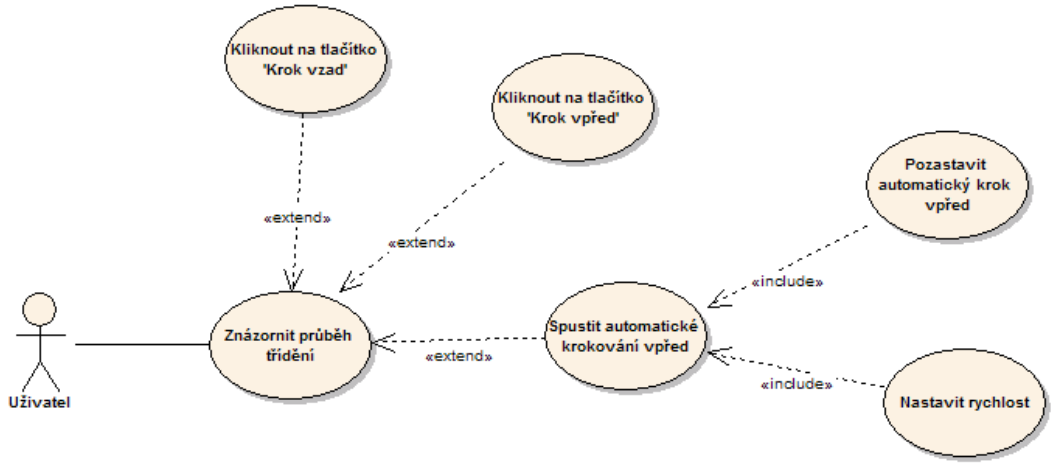
Obrázek 18. Základní use case diagram.

Možnosti zadávání čísel ke třídění jsou znázorněny na obrázku 19.



Obrázek 19. Use case diagram - zadávání čísel.

Poslední use case diagram (obr. 20.) zachycuje, jak lze znázornit průběh třídění.



Obrázek 20. Use case diagram - znázornění průběhu třídění.

4.2. Popis tříd

V této podkapitole si popíšeme jednotlivé třídy.

4.2.1. FormHlavni.cs

Hlavní třída celého projektu. Mezi její nejdůležitější metody a události patří:

- `public static int[] ExtrakceCelychCisel(string retezec)`
  extrahuje ze zadaného řetězce celá čísla, která vrací v poli.

- `public void GenerujCisla()`
  podle zvoleného počtu generovaných čísel vytvoří velké pole a naplní ho náhodně vygenerovanými čísly od 0 do 99.

- `private void NastavTlacitka(bool vpred = true)`
  parametr *vpred* určuje, zda se má proměnná *sortStepIndex* zvýšit nebo snížit o hodnotu 1. Tato metoda se volá v každém kroku třídění. Je vytvořena jako samostatná metoda kvůli aktualizaci ovládacích prvků (nastavuje, kdy mají být tlačítka *Krok vzad*, *Krok vpřed* a *Autom. vpřed* aktivní, aktualizuje údaj aktuální krok/celkový počet kroků třídění).

- `private void Vykresleni()`
  graficky zobrazuje průběh třídění. Prvky v poli jsou vykresleny jako čtverce (rectangle) s hodnotou prvku uprostřed. Pokud má prvek nastavenou určitou vlastnost, je výplň čtverce vykreslena danou barvou. Kvůli rozdílnosti třídících metod nelze tuto metodu napsat úplně obecně, jednotlivé případy jsou pak řešeny podmínkami.

- `private void TrideniUkonceno()`
  zastaví Timer, pokud je spuštěn a vyvolá formulář *FormOznameni*.

- `private void NoveTrideni()`
  smaže PictureBox, *sortStepList*, Labely, nastaví proměnou *sortStepIndex* na hodnotu 0 a zastaví Timer pokud je spuštěn.

- `private void btnKonec_Click(object sender, EventArgs e)`
  vyvolá formulář *FormDotaz*.

- `private void btnAlgoritmy_Click(object sender, EventArgs e)`
  vyvolá formulář *FormAlgoritmy*.

- `private void btnSpustTrideni_Click(object sender, EventArgs e)`
  do proměnné *zaznam* uloží obsah textového pole, pokud je textové pole prázdné nebo čísla nejsou korektně zadaná, vyvolá formulář *FormUpozorneni* s textem výjimky. Jinak je zavolaná metoda *RunSortMethod(bool vzes, int[] pole, ref List<SortStep> stepList)* ve třídě vybraného algoritmu.

4.2.2. SortStep.cs

Objekt *SortStep* představuje jednotlivý krok třídění, který má při vytvoření na vstupu pole čísel a aktivní (vybraný index). Definovány jsou tu také vlastnosti objektu jako srovnávaný prvek, přesunutý prvek, vkládaný prvek, pivot, levý a pravý následník, prvek na místě, setříděné prvky, neaktivní prvky a další.

4.2.3. Třídy algoritmů

V aplikaci je implementováno těchto 8 tříd algoritmů: *InsertSort.cs*, *BinaryInsertSort.cs*, *ShellSort.cs*, *BubbleSort.cs*, *ShakerSort.cs*, *QuickSort.cs*, *SelectSort.cs* a *HeapSort.cs*.

Při průchodu algoritmem se vytvářejí objekty (třída *SortStep.cs*) a skládají do pole (*stepList*). Je tak zaznamenán průběh třídění, tzn., že se vytváří každý krok třídění. Celé to pak slouží jako podklad pro vizualizaci (metoda *Vykresleni()*).

4.2.4. FormAgoritmy.cs

V této třídě jsou textové soubory s popisem algoritmů načteny přes *StreamReader* a následně zobrazeny v Labelu na příslušné záložce (TabPage) v TabControl.

4.2.5. FormDotaz.cs, FormOznameni.cs a FormUpozorneni.cs

Každá tato třída reprezentuje formulář, který byl vytvořen místo MessageBoxu. MessageBox nebyl použit, protože u něj nejde nastavit barva pozadí jako byla zvolena pro hlavní formulář aplikace.

4.3. Nápověda

Pro tvorbu nápovědy byl použit volně šířitelný software *HelpMaker 7.4* [12] s integrovaným WYSIWYG editorem. Zpracovaná nápověda byla zkompilovaná jako HTML nápověda (formát CHM).

4.4. Instalátor

Instalátor aplikace byl vytvořen nástrojem *Inno Setup 5.4.2* [13], který podporuje všechny verze Windows. Inno Setup komprimuje všechny soubory do jednoho spustitelného souboru EXE. Šířen je pod GPL licencí.

Závěr

Cílem této bakalářské práce bylo sestavit aplikaci, která demonstruje činnost algoritmů vnitřního třídění. Tohoto cíle bylo dosaženo. Snažila jsem se vytvořit uživatelsky přívětivou a snadno ovladatelnou aplikaci.

V aplikaci lze průběh třídění u jednotlivých algoritmů vnitřního třídění krokovat nebo nechat automaticky krokovat vpřed. Čísla ke třídění se mohou zadat vstupem z klávesnice nebo nechat náhodně vygenerovat. Posloupnost čísel ke třídění se dá setřídit vzestupně nebo sestupně. Aplikace také nabízí možnost vyvolat okno s popisem algoritmů třídění.

Aplikace by mohla být rozšířena ještě o další třídící algoritmy nebo lokalizovaná do anglického jazyka.

Při tvorbě této bakalářské práce jsem si rozšířila znalosti nejen o algoritmech třídění, ale také v programování v jazyce C#.

Reference

[1] SEDGEWICK, Robert. *Algorithms in C++, Parts 1-4, Fundamentals, Data structures, Sorting, Searching.* Addison-Wesley, 2004. ISBN 0-201-35088-2.

[2] TÖPFER, Pavel. *Algoritmy a programovací techniky.* Praha: Prometheus, 1995. ISBN 80-85849-83-6.

[3] VEČERKA, Arnošt. *Základní algoritmy.* Učební text, Olomouc, 2007.

[4] VIRIUS, Miroslav. *Základy algoritmizace.* Praha: ČVUT, 2008. ISBN 978-80-01-04003-4.

[5] WIRTH, Niklaus. *Algoritmy a štruktúry údajov.* Bratislva: Alfa, 1989. ISBN 80-05-00153-3.

[6] ČERNÝ, Jakub. *Základní grafové algoritmy* [online]. Verze 0.85. Dostupné z: http://kam.mff.cuni.cz/˜kuba/ka/ka.pdf [citováno 8. února 2011].

[7] DVORSKÝ, Jiří. *Algoritmy I.* [online]. Verze ze dne 28. února 2007. Dostupné z: http://www.cs.vsb.cz/dvorsky/Opora.html [citováno 10. února 2011].

[8] HRUBINA, J., TAUFER, I., TAUFER, J. *Algoritmy a algoritmizace: vývojové diagramy, sbírka řešených příkladů* [online]. Pardubice: Univerzita Pardubice, 2001. Kapitola 2. Dostupné z: http://krpvt.upce.cz/files/algoritmy/kap2.pdf [citováno 21. února 2011].

[9] PŘIKRYL, J., VLČEK, M. *Základy algoritmizace, Turingův stroj* [online]. 2. přednáška K611MA. Verze 2009-10-15 00:49. Dostupné z: http://euler.fd.cvut.cz/predmety/ma/files/ma-02-2009-slides.pdf [citováno 18. února 2011].

[10] *Al-Chorezmí - Wikipedie* [online].
Dostupné z: http://cs.wikipedia.org/wiki/Al-Khwarizmi [citováno 8. února 2011].

[11] *Microsoft Visual C# 2010 Express.*
Ke stažení z: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/.

[12] *HelpMaker 7.4.* Ke stažení z: http://helpmaker.en.softonic.com/.

[13] *Inno Setup Compiler 5.4.2.* Ke stažení z: http://www.jrsoftware.org/.

A. Obsah přiloženého CD

Na přiloženém CD naleznete následující adresáře a textový soubor:

`bin/`
: V tomto adresáři se nachází instalátor aplikace `VATsetup.exe` a adresář *Vizualizace algoritmů třídění*, z něhož se dá program spustit přímo z CD souborem `Vizualizace algoritmů třídění.exe`.

`doc/`
: Tento adresář obsahuje dokumentaci práce ve formátu PDF vytvořenou dle závazného stylu KI PřF, včetně vložených obrázků a zdrojové kódu práce ve formátu LaTeX.

`src/`
: Zde jsou kompletní zdrojové texty programu `Vizualizace algoritmů třídění`.

`install/`
: V adresáři je přiložen instalační soubor `dotNet40setup.exe` pro instalaci běhového prostředí Microsoft .NET Framework 4.0.

`readme.txt`
: V textovém souboru jsou uvedeny instrukce pro spuštění programu, včetně požadavků pro jeho provoz.